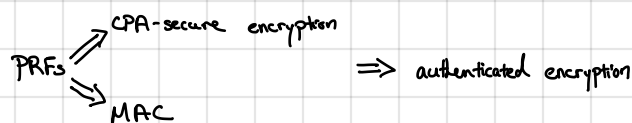


Thus far, we have assumed that parties have a shared key. Where does the shared key come from?

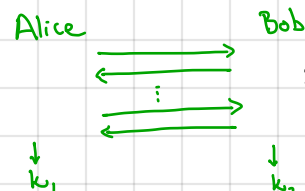
Can we do this using the tools we have developed so far?

So far in this course:



Can we use PRFs to construct secure key-agreement protocols?

Key agreement:



- Requirements:
- 1) $k_1 = k_2 = k$ with high probability
 - 2) Eavesdropper cannot learn k (efficiently)

Merkle puzzles: Suppose $f: X \rightarrow Y$ is a function that is hard to invert

("one-way function")

$\hookrightarrow$ for example, a secure PRG

$G: \{0,1\}^n \rightarrow \{0,1\}^n$ is one-way

Alice
 $x_1, \dots, x_n \leftarrow X$
 $y_i = f(x_i), \dots, y_n = f(x_n)$

Bob

$i \xleftarrow{R} [n]$

find x_i such that $f(x_i) = y_i$

derive a key k from x_i

[solve the "puzzle"]

$\uparrow$ we assume that the solution is unique

$\xleftarrow{\text{Encrypt}_{AE}(k, m)}$

$\downarrow$
 try each key k_i to decrypt ciphertext
 derived from x_i

Suppose it takes time t to solve a puzzle. Adversary needs time $O(nt)$ to solve all puzzles and identify key. Honest parties work in time $O(n+t)$.

$\hookrightarrow$ Only provides linear gap between honest parties and adversary

Can we get a super-polynomial gap just using PRGs?

Can we get a super-linear gap just using PRGs?

Very difficult! [Impagliazzo-Rudich]

Very difficult! [Barak-Mahmoody]

$\swarrow$ result holds even if start with a one-way permutation

Impagliazzo-Rudich: Proving the existence of key-agreement that makes black-box use of PRG implies $P \neq NP$.

We will turn to algebra/number theory for new sources of hardness to build key agreement protocols.

Definition. A group consists of a set G together with an operation $*$ that satisfies the following properties:

- Closure: If $g_1, g_2 \in G$, then $g_1 * g_2 \in G$
- Associativity: For all $g_1, g_2, g_3 \in G$, $g_1 * (g_2 * g_3) = (g_1 * g_2) * g_3$
- Identity: There exists an element $e \in G$ such that $e * g = g = g * e$ for all $g \in G$
- Inverse: For every element $g \in G$, there exists an element $g^{-1} \in G$ such that $g * g^{-1} = e = g^{-1} * g$

In addition, we say a group is commutative (or abelian) if the following property also holds:

- Commutative: For all $g_1, g_2 \in G$, $g_1 * g_2 = g_2 * g_1$

Notation: Typically, we will use " $\cdot$ " to denote the group operation (unless explicitly specified otherwise). We will write g^x to denote $\underbrace{g \cdot g \cdot g \cdots g}_{x \text{ times}}$ (the usual exponential notation). We use " 1 " to denote the multiplicative identity ↙ called "multiplicative" notation

Examples of groups: $(\mathbb{R}, +)$: real numbers under addition

$(\mathbb{Z}, +)$: integers under addition

$(\mathbb{Z}_p, +)$: integers modulo p under addition [sometimes written as $\mathbb{Z}/p\mathbb{Z}$]

The structure of $\mathbb{Z}_p^*$ (an important group for cryptography): ↙ here, p is prime

$$\mathbb{Z}_p^* = \{x \in \mathbb{Z}_p : \text{there exists } y \in \mathbb{Z}_p \text{ where } xy = 1 \pmod{p}\}$$

↑ the set of elements with multiplicative inverses modulo p

What are the elements in $\mathbb{Z}_p^*$?

Bezout's identity: For all positive integers $x, y \in \mathbb{Z}$, there exists integers $a, b \in \mathbb{Z}$ such that $ax + by = \gcd(x, y)$.

Corollary: For prime p , $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$.

Proof. Take any $x \in \{1, 2, \dots, p-1\}$. By Bezout's identity, $\gcd(x, p) = 1$ so there exists integers $a, b \in \mathbb{Z}$ where $1 = ax + bp$.
Modulo p , this is $ax = 1 \pmod{p}$ so $a = x^{-1} \pmod{p}$.

→ greatest common divisor

Coefficients a, b in Bezout's identity can be efficiently computed using the extended Euclidean algorithm:

Euclidean algorithm: algorithm for computing $\gcd(a, b)$ for positive integers $a > b$:

relies on fact that $\gcd(a, b) = \gcd(b, a \pmod{b})$:

to see this: take any $a > b$

↳ we can write $a = b \cdot q + r$ where $q \geq 1$ is the quotient and $0 \leq r < b$ is the remainder

↳ d divides a and $b \iff d$ divides b and r

↳ $\gcd(a, b) = \gcd(b, r) = \gcd(b, a \pmod{b})$

gives an explicit algorithm for computing $\gcd$: repeatedly divide:

$$\begin{array}{lll} \gcd(60, 27): & 60 = 27(2) + 6 & [q=2, r=6] \rightsquigarrow \gcd(60, 27) = \gcd(27, 6) \\ & 27 = 6(4) + 3 & [q=4, r=3] \rightsquigarrow \gcd(27, 6) = \gcd(6, 3) \\ & 6 = 3(2) + 0 & [q=2, r=0] \rightsquigarrow \gcd(6, 3) = \gcd(3, 0) = 3 \end{array}$$

"rewind" to recover coefficients in Bezout's identity:

$$\begin{array}{l} \text{extended Euclidean algorithm} \left\{ \begin{array}{l} 60 = 27(2) + 6 \\ 27 = 6(4) + 3 \\ 6 = 3(2) + 0 \end{array} \right. \rightarrow 3 = 27 - 6 \cdot 4 \end{array} \quad \begin{array}{l} 6 = 60 - 27(2) \\ \downarrow \\ 27 - (60 - 27(2))4 \\ = 27(9) + 60(-4) \end{array}$$

↑
coefficients

Iterations needed: $O(\log a)$ — i.e., bitlength of the input [worst case inputs: Fibonacci numbers]

Implication: Euclidean algorithm can be used to compute modular inverses (faster algorithms also exist)